APPENDIX A.II

Source code file named fac.pl.

```
%*******************************************************************************
%   File     : fac.pl
%   Primary Authors  : Adam Cheyer, David Martin
%   Purpose :  Provides communications and coordination of the activities
%            of a dynamic collection of client agents.
%   Updated : 12/98
%
%   ------------------------------------------------------------------------
%   Unpublished-rights reserved under the copyright laws of the United States.
%
%
%   Unpublished Copyright (c) 1998, SRI International.
%   "Open Agent Architecture" and "OAA" are Trademarks of SRI International.
%   ------------------------------------------------------------------------
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% fac.pl : the facilitator agent          Adam Cheyer
%                                               David Martin
%
% Provides communications and coordination of the activities of a
% dynamic collection of client agents.
%
% The blackboard can respond to the following external requests:
%
%     ev_post_event(AgentID, Cmd)    : sends event to the agent
%     ev_post_event(Cmd)       : sends event to all
%     ev_post_declare(Mode, Solvables, Params)
%                                : adds, removes or replaces solvables ON
%                                : the facilitator
%     ev_post_update(Mode, Clause, Params)
%                                : adds, removes, or replaces data
%                                  on appropriate agents
%     ev_post_trigger_update(Mode,TriggerType,Condition,Action,Params)
%                                : adds or removes a trigger
%                                  on appropriate agents
%     ev_post_solve(Goal, Params): finds agent(s) to solve Goal
%     connected(Connection)    : records that a client agent has connected
%       ev_connect(AgentInfo)
%                                : additional information from a client
%                                : agent (having version > 3.0)
%     end_of_file(Connection) : records that a client has closed its
%                          connection
%     ev_register_solvables : records the goals that an agent can solve.
%
% A facilitator uses the following events internally as trigger actions:
%
%     ev_respond_query(Id,ToKS,ByKS,G,OrigParams,Params,S)
%                                : Sends the result of a query back to KS
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- use_module(library(lists)).
:- use_module(library(basics)).
:- use_module(library(strings)).
:- use_module(library(charsio)).
```

```
:- use_module(library(sets)).
:- use_module(library(samsort)).  % for samsort(Ordered,Raw,Sort)
:- use_module(library(tcp), [tcp_now/1, tcp_time_plus/3,
                        tcp_schedule_wakeup/2, tcp_cancel_wakeup/2]).


% The file containing the com module is normally specified here.  For
% more info, see comments near the top of oaa.pl.
:- use_module(com_tcp, all).
:- use_module(oaa, all).

% Whether or not to load translations and compound query code
% is determined right here:
% :- [compound].
:- [translations].


:- multifile oaa_AppDoEvent/2.

:- dynamic time_limit_trigger/5.   % time_limit_trigger(Id,When,KS,Goal,Params)
:- dynamic goal_count/10.        % goal_count(GoalId,Goal,Params,EvParams,
                                   %   ToBeCalled,Called,Responders,Solvers,
                                   %   Answers,NumAnswers)
:- dynamic update_count/4.       % update_count(GoalId,NumAgentsRequested,
                            %   KSs, Updaters)
initial_solvables([
  solvable(agent_data(_Id, _Status, _Solvables, _Name), [type(data)],
                                                    [write(true)]),
    % Locations of all facilitators (currently maintained only by the 'root'
    % facilitator:
  solvable(agent_location(_Id2, _Name2, _Host2, _Port2), [type(data)],
                                                      [write(true)]),
    % Host (if known) of each client agent:
  solvable(agent_host(_Id3, _Name3, _Host3), [type(data)], [write(true)]),
  agent_version(_Id1, _Language1, _Version1),
  can_solve(_Goal4, _IdList4),
    %  For backwards compatibility.  In translations.pl, some events
    %  (write_bb, etc.) specify updates to this solvable.  Also, old-style
    % data triggers refer to it:
  solvable(data(_Item, _Data), [type(data)], [write(true)])
]).

/* Agent specific declarations */

oaa_AppInit :-
    oaa_SetTimeout(0).

/* This is the event generated by the TCP library.  Will be followed
   immediately by ev_connect/4, which is constructed by the client agent */
oaa_AppDoEvent(connected(Connection), _) :-
        !,
      format('~nKnowledge source connected: ~p~n~n', [Connection]),
      Id = Connection,
      oaa:oaa_add_data_local(agent_data(Id, open, [], Id), []),
      %% Maintain information of currently connected data.
      add_connected(Id, Connection).
```

```
/* For now, the ID of a client agent is the same as its connection (socket).
   This could change in the future, so we store Id and Connection
   as two separate entities. */
oaa_AppDoEvent(ev_connect(AgentInfoList), Params) :-
        memberchk( connection_id(Id), Params),
      oaa_Name(MyName),
      oaa_Id(MyId),
      MyLanguage = prolog,
      oaa_LibraryVersion(MyVersion),

      update_connected(Id, AgentInfoList),

      % preferred TCP transfer mechanism
      MyFormat = quintus_binary,

      % Inform the client of his Id, and info about me.
        com_SendData(Id,
         event(ev_connected([oaa_id(Id), fac_id(MyId), fac_name(MyName),
           fac_lang(MyLanguage), fac_version(MyVersion),
           format(MyFormat)]),
             [])).

/* Removes meta-data for KS when the KS deconnects */
oaa_AppDoEvent(end_of_file(Connection), _) :-
        Id = Connection,
      remove_connected(Id),
      oaa:oaa_remove_data_local(agent_data(Id, _Status, _Solvable, AgentName),
[]),
      format('~nKnowledge source disconnected: ~p (~p)~n~n', [Id,AgentName]),
        % remove all facts written by the agent
      % TBD: Is this getting all relevant triggers (see commented code below)?
        oaa:oaa_remove_data_owned_by(Id),

% Do we really want to do this?  I think clients who are interested could
% register a trigger on the agent_data predicate.
% Rather, I think we should check to see if any agents are currently waiting
% for this agent to solve some goal -- if the agent disconnects, we can assume
% that it won't be solving the goal anytime soon, and we should send back
% failure to the requesting agent.  See OAA 1.0 Facilitator, end_of_file()
% method.  [AJC, 11/24/97]
      post_to_all_clients(ev_agent_disconnected(Id)).

%     fail.
% TBD: This needs update to look at the persistence param.
% oaa_AppDoEvent(end_of_file(KS), _) :-
%     % remove all triggers for KS
%     on_exception(_, trigger(KS, Type, Kind, OpMask, Template, Cond, Action),
fail),
%     retract(trigger(KS, Type, Kind, OpMask, Template, Cond, Action)),
%     fail.
% oaa_AppDoEvent(end_of_file(_KS), _) :- !.

oaa_AppDoEvent(ev_ready(Name), Params) :-
        memberchk(from(Id), Params),
      % TBD: Let's have an error message if this fails:
      oaa:oaa_remove_data_local(agent_data(Id, _OldStatus, Solvables, _Name),
Params),
```

```
      oaa:oaa_add_data_local(agent_data(Id, ready, Solvables, Name), Params).

/* Stores the goals that a KS knows how to solve */
% Is this obsolete?
oaa_AppDoEvent(ev_register_solvables(Goals), Params) :-
        memberchk(from(KS), Params),
      oaa_AppDoEvent(ev_register_solvables(add,Goals,KS,[]), Params), !.

    % IMPORTANT: We assume the Solvables are in standard form and can
    % legally be added/removed/replaced for this agent.  Also, we take
    % care to keep the facilitator's copy of each client's solvables
    % identical to that stored at the client.  (Compare to code in
    % liboaa.pl, pred. oaa_declare_local).
oaa_AppDoEvent(ev_register_solvables(Mode,Solvs,AgentName,EvParams), Params) :-
        memberchk(from(KS), Params),
      oaa_Name(KSName),
      (oaa:oaa_remove_data_local(agent_data(KS, Status, List, _AgentName),
Params)
         ;
       format('STRANGE! register_solvables called by unknown KS!!!: ~p~n',
            [KS]),
       Status = ready,
       List = []
      ),
      icl_ConvertSolvables(PrettySolvs, Solvs),
      ( Mode == add, memberchk(if_exists(overwrite), EvParams) ->
          NewList = Solvs,
          format('~p (~p) can solve: ~n  ~p~n~n', [KS, AgentName,
                                          PrettySolvs])
      | Mode == add ->
          append(List, Solvs, NewList),
          format('~p (~p) has added solvables: ~n  ~p~n~n',
               [KS, AgentName, PrettySolvs])
      | Mode == remove ->
          subtract(List, Solvs, NewList),
          format('~p (~p) has removed solvables: ~n  ~p~n~n',
               [KS, AgentName, PrettySolvs])
      | Mode == replace ->
          memberchk(with(NewSolvable), EvParams),
          Solvs = [Solvable],
          oaa:replace_element(Solvable, List, NewSolvable, NewList),
          format('~p (~p) has replaced solvable:~n  ~p~nwith solvable:~n
~p~n~n',
               [KS, AgentName, Solvable, NewSolvable])
      ),
      oaa:oaa_add_data_local(agent_data(KS, Status, NewList, AgentName),
Params),

      % if a parent exists (not root), pass goals upward.
      (com:com_GetInfo(parent, connection(_C)) ->
         oaa_PostEvent( ev_register_solvables(Mode,Solvs,EvParams,KSName),
                        [address(parent)])
      |
         true),
      !.
```

```
/* A client has requested that I declare certain solvables.
   TBD: This is still sketchy; should include some validation of the
   request, and should ensure the perms and params are right. */
oaa_AppDoEvent(ev_post_declare(Mode, Solvables, Params), EvParams) :-
    memberchk(from(RequestingKS), EvParams),
    oaa:oaa_declare_local(Mode, Solvables, Params, NewSolvables),
    icl_ConvertSolvables(PrettySolvs, NewSolvables),
    oaa_Id(MyId),
    oaa_Name(MyName),
    format('~p (~p) has added solvables: ~n  ~p~n~n',
         [MyId, MyName, PrettySolvs]),
    oaa_PostEvent(
             ev_reply_declared(Mode, Solvables,Params, NewSolvables),
             [address(RequestingKS)]).

% A client requests a data solvable update operation (add, remove, replace)
% on the appropriate agents.
oaa_AppDoEvent(ev_post_update(Mode, Clause, Params), EvParams) :-
    ( Clause = (Head :- _Body) ->
      true
    | otherwise ->
      Head = Clause
    ),
    memberchk(from(RequestingKS), EvParams),
    % see if the query is addressed using address(KS) in Params
    check_address(Params, AddrKS),
    choose_agents_for_data(RequestingKS,Head,AddrKS,write,false,KSList),
    dispatch_update_request(RequestingKS, Mode, Clause, Params, KSList).

% A client requests a trigger update operation (Mode = add, remove, replace)
% on the appropriate agents.  For triggers of type comm' and time', the
% address parameter must be present (otherwise, the request should not
% have come to the facilitator). For the other types, the address is
% optional.

oaa_AppDoEvent(ev_post_trigger_update(Mode, data, Condition,
                              Action, Params), EvParams) :-
    !,
    memberchk(from(RequestingKS), EvParams),
    % see if the query is addressed using address(KS) in Params
    check_address(Params, AddrKS),
    choose_agents_for_data(RequestingKS,Condition,AddrKS,call,false,KSList),
    append(Params, EvParams, AllParams),
    dispatch_trigger_request(RequestingKS, Mode, data, Condition, Action,
                             AllParams, KSList).
oaa_AppDoEvent(ev_post_trigger_update(Mode, task, Condition,
                              Action, Params), EvParams) :-
    !,
    memberchk(from(RequestingKS), EvParams),
    % see if the query is addressed using address(KS) in Params
    check_address(Params, AddrKS),
    choose_agents_for_goal(RequestingKS,Condition,AddrKS,Params,false,KSList),
    append(Params, EvParams, AllParams),
    dispatch_trigger_request(RequestingKS, Mode, task, Condition, Action,
                             AllParams, KSList).

oaa_AppDoEvent(ev_post_trigger_update(Mode, Type, Condition,
```

```
                                Action, Params), EvParams) :-
    memberchk(from(RequestingKS), EvParams),
    check_address(Params, KSList),
    is_list(KSList),
    append(Params, EvParams, AllParams),
    dispatch_trigger_request(RequestingKS, Mode, Type, Condition, Action,
                        AllParams, KSList).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TBD: New for compound goals:

% If satisfaction of a compound goal is requested, and the compound query
% interpreter is not included, signal error condition:
oaa_AppDoEvent(ev_post_solve(Goal, Params), EvParams) :-
    \+ current_predicate(complete_goal, complete_goal(_,_,_,_)),
    \+ icl_BasicGoal(Goal),
    !,
    format('ERROR: This facilitator does not support compound goals~n', []),
    format('  Returning 0 solutions for goal:~n  ~w~n', [Goal]),
    oaa_Id(Facilitator),
    memberchk(from(RequestingKS), EvParams),
    oaa_PostEvent(
             ev_reply_solved([Facilitator],[],Goal,Params,[]),
[address(RequestingKS)]).

% If compound goal capabilities are included, ALL ev_post_solve events are
% handled here.  Otherwise, they fall through to later clauses.
oaa_AppDoEvent(ev_post_solve(Goal, Params), EvParams) :-
    current_predicate(complete_goal, complete_goal(_,_,_,_)),
    !,
    memberchk(from(RequestingKS), EvParams),
    complete_goal(RequestingKS, Goal, Params, CompletedGoal),
    execute_goal(RequestingKS, Goal, Params, CompletedGoal).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


/* Finds all KSs for a goal, asks them to solve it, then returns */
/* the answers to the calling KS                                 */
oaa_AppDoEvent(ev_post_solve(Goal, Params), EvParams) :-
        memberchk(from(RequestingKS), EvParams),
      % see if the query is addressed using address(KS) in Params
      check_address(Params, AddrKS),
        choose_agents_for_goal(RequestingKS,Goal,AddrKS,Params,true,KSList),

      % if none of my agents know how to solve goal, send to parent
      (KSList = [] ->

         find_level(Params, Level, NewParams),
         ((com:com_GetInfo(parent, fac_name(ParentName)),
           Level > 0) ->
              oaa_TraceMsg('~nRouting goal "ev_solve(~p)" to parent ~p.~n',
           [Goal, ParentName]),

           new_goal_id(Id),
           oaa_PostEvent( ev_post_solve_from_bb(Id, Goal, NewParams),
                          [address(parent)]),
```

```
            % if answers requested,
            % send parent's answers directly back to requestingKS
            % as well as blackboard solutions
            (memberchk(reply(none), NewParams) -> true |
             % No longer valid:
             % send_blackboard_solutions(RequestingKS, Goal, Params),
             oaa:oaa_add_trigger_local(
                 comm,
               event(ev_reply_solved_by_bb(Id,SomeKS,Goal,Params2,Solutions),
                   _),
             ev_respond_query(Id,RequestingKS,SomeKS,Goal,Params,Params2,
                         Solutions),
             [recurrence(when), on(receive)])
            )
          |
             % root blackboard: doesn't know anyone who can solve goal
            (memberchk(reply(none), NewParams) -> true |
               oaa_Id(KSID),
               oaa_PostEvent(
                ev_reply_solved([KSID],[],Goal,Params,[]),
                [address(RequestingKS)])
            )
        )

      | otherwise ->
          dispatch_solve_request(RequestingKS, Goal, Params, EvParams, KSList)
      ).

/* Finds all KSs for a goal, asks them to solve it, then returns */
/* the answers to the calling BB                                 */
oaa_AppDoEvent(ev_post_solve_from_bb(Id, Goal, Params), EvParams) :-
        memberchk(from(RequestingKS), EvParams),
      % see if the query is addressed using address(KS) in Params
      check_address(Params, AddrKS),
      choose_agents_for_goal(RequestingKS,Goal,AddrKS,Params,true,KSList),

      % if none of my agents know how to solve goal, send to parent
      (KSList = [] ->

         find_level(Params, Level, NewParams),
         % try to ask parent
         ((com:com_GetInfo(parent, fac_name(ParentName)),
           com:com_GetInfo(parent, fac_id(ParentId)), Level > 0) ->
              oaa_TraceMsg('~nRouting goal "ev_solve(~p)" to parent ~p.~n',
            [Goal, ParentName]),

            oaa_PostEvent( ev_post_solve_from_bb(Id, Goal, NewParams),
                           [address(parent)]),

            (memberchk(reply(none), NewParams) -> true |
              oaa:oaa_add_trigger_local(
                 comm,
               event(ev_reply_solved_by_bb(Id, _SomeKS, Goal, P2, Solutions),
                   _),
               ev_respond_bb_query(RequestingKS,ParentId,Id,Goal,Params,
                           P2, Solutions),
```

```
            [recurrence(when), on(receive)])
          )
        |
          % root blackboard : knows no solvers
          (memberchk(reply(none), Params) -> true |
            oaa_Name(KSName),
            oaa_PostEvent(
             ev_reply_solved_by_bb(Id, KSName,Goal,Params, []),
             [address(RequestingKS)])
          )
        )

      |
        member(SomeKS, KSList),    % backtrack over all KSs.
          oaa_TraceMsg('~nRouting goal to ~p: ~p~n',
                       [SomeKS, Goal]),

        oaa_PostEvent( ev_solve(Id, Goal, Params),
                       [address(SomeKS), from(RequestingKS)]),
        (memberchk(reply(none), Params) -> fail |
          oaa:oaa_add_trigger_local(
            comm,
            event(ev_solved(Id, _SomeKS, Goal, P2, Solutions), _),
            ev_respond_bb_or_post_higher(RequestingKS,SomeKS,Id,
            Goal,P2,Solutions),
           [recurrence(when), on(receive)])
        ),
        fail     % send events to all KSs that can solve goal.
      ).


oaa_AppDoEvent(wakeup(time_limit(Id)), _EvParams) :-
      retract(time_limit_trigger(Id,_When,RequestingKS,Goal,Params)),
      oaa_TraceMsg('~nTime limit expired. Goal failed:~n   ~p~n', [Goal]),
      oaa_Id(KSId),     % get local ksid
%      interpret(KSId,
%        ev_respond_query(-1,RequestingKS, KSId, Goal, Params, Params, [])).
      oaa_Interpret(
          ev_respond_query(-1,RequestingKS, KSId, Goal, Params, Params, []),
        [from(KSId)]).


% When asked by parent blackboard to solve a goal,
% route all answers back using "ev_solved(Id, KS, Goal, Params, Solutions)".
oaa_AppDoEvent(ev_solve(Id, Goal, Params), EvParams) :-
        memberchk(from(ParentBB), EvParams),
      oaa_Name(KSName),

      % see if the query is addressed using address(KS) in Params
      check_address(Params, AddrKS),
      choose_agents_for_goal(KSName,Goal,AddrKS,Params,true,KSList),

      % if none of my agents know how to solve goal, send empty solutions
      (KSList = [] ->
         (memberchk(reply(none), Params) -> true |
           oaa_PostEvent( ev_solved(Id,KSName,Goal,Params,[]),
                          [address(ParentBB)])
```

```
        )
      |
        member(SomeKS, KSList),     % backtrack over all KSs.
          oaa_TraceMsg('~nRouting goal "ev_solve(~p)" to ~p.~n', [Goal,
SomeKS]),

        oaa_PostEvent( ev_solve(Id, Goal, Params),
                       [address(SomeKS), from(ParentBB)]),
        (memberchk(reply(none), Params) -> fail |
          oaa:oaa_add_trigger_local(
             comm,
             event(ev_solved(Id, _SomeKS, Goal, P2, Solutions), _),
             ev_respond_to_parent(ParentBB,KSName,Id,Goal,Params,
                                  P2, Solutions),
           [recurrence(when), on(receive)])
        ),
        fail     % send events to all KSs that can solve goal.
      ).

/* If a KS is available, send it the message */
oaa_AppDoEvent(ev_post_event(Event), EvParams) :-
        memberchk(from(KS), EvParams),
        choose_ks_for_goal(KS, Event, _, [], SomeKS, _),
      oaa_PostEvent(Event, [address(SomeKS), from(KS)]),
      fail.

/* If a KS is available, send it the message */
oaa_AppDoEvent(ev_post_event(KSName, Event), EvParams) :-
      oaa_Name(KSName), !,
      % interpret(KS, Event).
      oaa_Interpret(Event, EvParams).
oaa_AppDoEvent(ev_post_event(KSName, Event), EvParams) :-
        memberchk(from(KS), EvParams),
        % agent must be "ready" to receive messages, or just
      % open if it is an agent compiled with old agentlib.
        (oaa:oaa_solve_local(agent_data(RealKS, ready, _Solvable,AgentName), [])
;
       oaa:oaa_solve_local(agent_data(RealKS, open, _Solvable,AgentName), []),
       oaa_Version(RealKS, _Language, Version),
       Version < 2.0),
      (match_ks(KSName, RealKS) ; KSName = AgentName),
      oaa_PostEvent(Event, [address(RealKS), from(KS)] ),
      fail.
% oaa_AppDoEvent(ev_post_event(_KS, _Event), _KS) :- !.
oaa_AppDoEvent(ev_post_event(_KS, _Event), _EvParams) :- !.



% Send back solutions to KS who originally requested them (with ev_post_solve)
%
% 970219: DLM: Added arg. OrigParams.  There is now a requirement that
% the params returned in a ev_reply_solved event must be unifiable with the
original
% params (from the corresponding solve event).
oaa_AppDoEvent(ev_respond_query(Id,RequestingKS, Requestee, Goal, OrigParams,
                      Params,Solutions), _EvParams) :-
        oaa_TraceMsg('~nRouting answers back to ~p:~n   ~p~n',
```

```
                [RequestingKS,Solutions]),
      cancel_time_check(Id),
      unify_params(OrigParams, Params, UParams),
      ( Solutions == [] ->
          Solvers = []
      | otherwise ->
          Solvers = [Requestee]
      ),
      oaa_PostEvent( ev_reply_solved([Requestee], Solvers, Goal, UParams,
Solutions),
                    [address(RequestingKS)]), !.

% Send back solutions to KS who originally requested them (with ev_post_solve)
% If no solutions, ask a higher blackboard
oaa_AppDoEvent(
   ev_respond_or_post_higher(RequestingKS, Solver,Id,Goal,P,Solutions),
   _EvParams) :-
      ((Solutions \== [] ; oaa:oaa_class(root)) ->
         cancel_time_check(Id), !,
         return_solutions(RequestingKS, Solver, Id, Goal,P,Solutions)
      |
         % @@DLM: The following needs work.  Must check goal_count status
         % before posting higher
         % sub-agents found no solutions: post higher
         com:com_GetInfo(parent, fac_id(ParentId)),
         find_level(P, Level, NewParams),
         Level > 0,
         oaa_PostEvent( ev_post_solve_from_bb(Id, Goal, NewParams),
                        [address(parent)]),
         oaa:oaa_add_trigger_local(
            comm,
            event(ev_reply_solved_by_bb(Id, _SomeKS, Goal, P2, Solutions),
                  _),
            ev_respond_query(Id,RequestingKS,ParentId,Goal,P,P2, Solutions),
            [recurrence(when), on(receive)])
      ).


% Send back acknowledgement to agent that originally requested an update.
oaa_AppDoEvent(
   ev_return_update(RequestingKS, Mode, Solver, Id, Clause, Params, Updaters),
   _EvParams) :-
      return_update(RequestingKS, Mode, Solver, Id, Clause, Params, Updaters).
% Send back acknowledgement to agent that originally requested a trigger
% update.
oaa_AppDoEvent(
   ev_return_trigger_update(RequestingKS, Mode, Solver, Id, Type, Condition,
                        Action, Params, Updaters),
   _EvParams) :-
        oaa_TraceMsg('~nRouting trigger updaters back to ~p:~n   ~p~n',
                [RequestingKS,Updaters]),
      return_trigger_update(RequestingKS, Mode, Solver, Id, Type, Condition,
                          Action, Params, Updaters).

% Send back solutions to a blackboard who requested them
%   (with ev_post_solve_from_bb)
%
```

```
% 970219: DLM: Added arg. OrigP.  There is now a requirement that
% the params returned in a ev_solved event must be unifiable with the original
% params (from the corresponding solve event).
oaa_AppDoEvent(ev_respond_bb_query(RequestingBB, Solver, Id,Goal,
                        OrigP, P,Solutions), _EvParams) :-
        unify_params(OrigP, P, UP),
        oaa_TraceMsg('~nRouting answers back to blackboard ~p:~n   ~p~n',
                [RequestingBB,Solutions]),
      oaa_PostEvent( ev_reply_solved_by_bb(Id,Solver,Goal,UP,Solutions),
                  [address(RequestingBB)]), !.

% Send back solutions to a blackboard who requested them
oaa_AppDoEvent(
    ev_respond_bb_or_post_higher(RequestingBB,Solver,Id,Goal,P,Solutions),
    _EvParams) :-
      ((Solutions \== [] ; oaa:oaa_class(root)) ->
           oaa_TraceMsg('~nRouting answers back to blackboard ~p:~n   ~p~n',
                [RequestingBB,Solutions]),
         oaa_PostEvent( ev_reply_solved_by_bb(Id, Solver, Goal, P,Solutions),
                    [address(RequestingBB)])
      |
         % sub-agents found no solutions: post higher
         com:com_GetInfo(parent, fac_id(ParentId)),
         find_level(P, Level, NewParams),
         Level > 0,
         oaa_PostEvent( ev_post_solve_from_bb(Id, Goal, NewParams),
                        [address(parent)]),
         oaa:oaa_add_trigger_local(
            comm,
            event(ev_reply_solved_by_bb(Id, _SomeKS, Goal, P2, Solutions),
                  _),
            ev_respond_bb_query(RequestingBB,ParentId,Id,Goal,P,P2,Solutions),
            [recurrence(when), on(receive)])
      ).


% Send back solutions to KS who originally requested them (with ev_post_solve)
%
% 970219: DLM: Added arg. OrigP.  There is now a requirement that
% the params returned in a ev_solved event must be unifiable with the original
% params (from the corresponding solve event).
oaa_AppDoEvent(ev_respond_to_parent(ParentBB,Solver,Id,Goal, OrigP,
                           P, Solutions), _EvParams) :-
        unify_params(OrigP, P, UP),
        oaa_TraceMsg('~nRouting answers back to parent bb ~p:~n   ~p~n',
                [ParentBB,Solutions]),
      oaa_PostEvent( ev_solved(Id, Solver, Goal, UP, Solutions),
                  [address(ParentBB)]), !.

oaa_AppDoEvent(ev_check_agent_name(KSName), EvParams):-
        memberchk(from(KS), EvParams),
        findall(KSName, oaa:oaa_solve_local(agent_location(_KSID, KSName ,_,_),
[]), L),
        (L==[] ->
         % @@tcp_send shouldn't be used:
        tcp_send(KS, 'UNIQUE');
        findall(KS1, oaa:oaa_solve_local(agent_location(_, KS1, _,_), []), R),
```

```
        tcp_send(KS, R)),!.

oaa_AppDoEvent(ev_register_port_number(Name,Address), EvParams) :- %+KS, +Port,
+Host
        memberchk(from(KS), EvParams),
        Address =.. [address, Port, Host],
        oaa:oaa_remove_data_local(agent_location(KS, _Name, _Port, _Host),
[]),!,
        oaa:oaa_add_data_local(agent_location(KS, Name, Port, Host), []),
        format('Agent ~p has Port: ~p , Host: ~p ~n', [KS, Port, Host]),
        !.

oaa_AppDoEvent(ev_register_port_number(Name,Address), EvParams) :- %+KS, +Port,
+Host
        memberchk(from(KS), EvParams),
        Address =.. [address, Port, Host],
        oaa:oaa_add_data_local(agent_location(KS, Name, Port, Host), []),
        format('Agent ~p has Port: ~p , Host: ~p ~n', [KS, Port, Host]),
        !.

oaa_AppDoEvent(ev_continue_execution(Id, RKS, Requestees, Solvers, Solutions),
             _EvParams) :-
        continue_execution(Id, RKS, Requestees, Solvers, Solutions).

% This is called from a trigger set in compound.pl.
oaa_AppDoEvent(
    ev_unify_and_continue_execution(Id, RKS, Goal, Vars, Requestee, Requestees,
Solvers, Solutions),
    _) :-
    unify_and_continue_execution(Id, RKS, Goal, Vars, Requestee, Requestees,
Solvers, Solutions).

/* Facilitator solvable: report the version and language of some
   connected agent. */
oaa_AppDoEvent(agent_version(Id, Language, Version), _EvParams) :-
    !,
    oaa_Version(Id, Language, Version).

/* Facilitator solvable: Find all agents who can solve goal */
oaa_AppDoEvent(can_solve(Goal, KSList), EvParams) :-
        ( memberchk(from(KS), EvParams) -> true | oaa_Id(KS) ),
      findall(SomeKS, choose_ks_for_goal(KS, Goal, _, [], SomeKS, _), KSList).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  % choose_agents_for_goal(RequestingKS,Goal,AddrKS,Params,Sort,Agents).
  %
  % The first 4 arguments are exactly as expected by choose_ks_for_goal.
  % Sort, a boolean, tells whether to sort on utility.
choose_agents_for_goal(RequestingKS,Goal,AddrKS,Params,Sort,Agents) :-
    findall(
      p(Agent,Utility),
      choose_ks_for_goal(RequestingKS,Goal,AddrKS,Params,Agent,Utility),
      Pairs
    ),
    ( Sort ->
      samsort(oaa_utility_compare, Pairs, SortedPairs)
    | otherwise ->
```

```
      SortedPairs = Pairs
    ),
    findall(Agent, member(p(Agent,_Utility), SortedPairs), Agents).

  % choose_agents_for_data(RequestingKS,Goal,AddrKS,Perm,Sort,Agents).
  %
  % The first 4 arguments are exactly as expected by choose_ks_for_data.
  % Sort, a boolean, tells whether to sort on utility.
choose_agents_for_data(RequestingKS,Goal,AddrKS,Perm,Sort,Agents) :-
    findall(
      p(Agent,Utility),
      choose_ks_for_data(RequestingKS,Goal,AddrKS,Perm,Agent,Utility),
      Pairs
    ),
    ( Sort ->
      samsort(oaa_utility_compare, Pairs, SortedPairs)
    | otherwise ->
      SortedPairs = Pairs
    ),
    findall(Agent, member(p(Agent,_Utility), SortedPairs), Agents).

oaa_utility_compare(p(_Agent1,Utility1), p(_Agent2,Utility2)) :-
    Utility1 >= Utility2.


/* Finds a KS that knows how to solve Goal */

% backtracks over all KSs that know how to solve
%   a particular goal, except for RequestingKS, which is the
%   KS who asked for the goal to be solved in the
%   first place.  (RequestingKS is included if the 'reflexive' Param
%   is present.)
% MemberList can be a list used to reduce the set to at most MemberList
%   or can be a specific KS to try, or a variable.
% If an address is specified in MemberList, it can be the same as
%   RequestingKS (DLM, 96/10/30).
% Solvable lists can contain complex tests (AC, 97/2/5)
%     e.g. [goal1(Y),(g(X) :- X > 1,X < 10),goal2]
% Params is now used to check for 'reflexive' (DLM, 97/03/06).
% Utility is the numeric value the KS has associated with the
%   solvable.
choose_ks_for_goal(RequestingKS, Goal, MemberList, Params, SomeKS, Utility) :-
       var(MemberList),
      !,
       ks_ready(SomeKS, ListOfGoals),
      ( icl_GetParamValue(reflexive(true), Params) ->
           true
      | otherwise ->
          SomeKS \== RequestingKS
      ),
      oaa:oaa_goal_matches_solvables(Goal, ListOfGoals, _, Matched),
      Matched = solvable(_, SolveParams, _),
      icl_GetParamValue(utility(Utility), SolveParams).
choose_ks_for_goal(_RequestingKS, Goal, MemberList, _Params, SomeKS, Utility) :-
      (is_list(MemberList) ->
         member(SomeKS, MemberList)
      | SomeKS = MemberList),
```

```
      oaa:icl_true_id(SomeKS, TrueId),
        ks_ready(TrueId, ListOfGoals),
      oaa:oaa_goal_matches_solvables(Goal, ListOfGoals, _, Matched),
      Matched = solvable(_, SolveParams, _),
      icl_GetParamValue(utility(Utility), SolveParams).

% backtracks over all KSs that know how to write a particular goal (or
% read, though that's not currently used), except for RequestingKS,
% which is the KS who asked for the goal to be solved in the first
% place.  RequestingKS is never included, because he does the
% appropriate asserts locally, when appropriate.
%
% Perm is 'read' or 'write'.

choose_ks_for_data(RequestingKS, Goal, MemberList, Perm, SomeKS, Utility) :-
        var(MemberList),
      !,
        ks_ready(SomeKS, ListOfGoals),
      SomeKS \== RequestingKS,
      oaa:oaa_data_matches_solvables(Goal, ListOfGoals, Perm, _, Matched),
      Matched = solvable(_, SolveParams, _),
      icl_GetParamValue(utility(Utility), SolveParams).
choose_ks_for_data(_RequestingKS, Goal, MemberList, Perm, SomeKS, Utility) :-
      (is_list(MemberList) ->
         member(SomeKS, MemberList)
      | SomeKS = MemberList),
        ks_ready(SomeKS, ListOfGoals),
      oaa:oaa_data_matches_solvables(Goal, ListOfGoals, Perm, _, Matched),
      Matched = solvable(_, SolveParams, _),
      icl_GetParamValue(utility(Utility), SolveParams).

% ks_ready(*SomeKS, *ListOfGoals).
%   Backtracks over all agents that are ready to solve goals.
%   If SomeKS is bound (with an agent's local ID), only that agent is
%   considered.
ks_ready(SomeKS, ListOfGoals) :-
        % agent must be "ready" to receive messages, or just
      % open if it is an agent compiled with old agentlib.
        (oaa:oaa_solve_local(agent_data(SomeKS, ready, ListOfGoals,_AgentName),
[]) ;
       oaa:oaa_solve_local(agent_data(SomeKS, open, ListOfGoals,_AgentName),
[]),
       oaa_Version(SomeKS, _Language, Version),
       Version < 2.0).
% Facilitator agents look up their own solvables in oaa_solvables/1.
ks_ready(SomeKS, ListOfGoals) :-
    oaa_Id(SomeKS),
    oaa:oaa_solvables(ListOfGoals).

match_ks(all, _KS).
match_ks(KS, KS).

% If params contains a VALID address (symbolic name or id) for one or more
% agents, return the agents' ids.
% If params contains an INVALID address, remove it from the list returned.
% Otherwise, KSAddr should return a variable.
% 97-05-23 (DLM): The address param now should always contain a list,
```

```
% but we'll check just to be safe.

check_address(Params, KSAddr) :-
      memberchk(address(Addr), Params),
      ( is_list(Addr) ->
          AddrList = Addr
      |   AddrList = [Addr]),
      find_addresses(AddrList, KSAddr),
      !.
check_address(_Params, _SomeKS).

find_addresses([], []).
find_addresses([Addr | Addrs], [Id | Ids]) :-
    find_address(Addr, Id),
    !,
    find_addresses(Addrs, Ids).
find_addresses([_Addr | Addrs], Ids) :-
    find_addresses(Addrs, Ids).

% Given an agent id (eg. 5) or a symbolic name (eg. 'interface')
% returns the local id for the reference.
%
% TBD: This does not yet handle remote addresses (associated with a different
% facilitator).

find_address(addr(Addr), SomeKS) :-
    com:com_GetInfo(incoming, oaa_addr(Addr)),
    % That's me, the facilitator.
    !,
    oaa_Id(SomeKS).
find_address(addr(Addr, SomeKS), SomeKS) :-
    com:com_GetInfo(incoming, oaa_addr(Addr)),
    % One of my clients.
    !,
    % Make sure it's current:
    oaa:oaa_solve_local(agent_data(SomeKS, _, _ListOfGoals, _AgentName), []).
find_address(name(Name), SomeKS) :-
    !,
    atom(Name),
    oaa:oaa_solve_local(agent_data(SomeKS, _, _ListOfGoals, Name), []).
find_address(SomeKS, SomeKS) :-
    oaa:oaa_solve_local(agent_data(SomeKS, _, _ListOfGoals, _AgentName), []),
    !.


find_level(Params, Level, NewParams) :-
      oaa:remove_element(level_limit(Level), Params, Params2), !,
      (Level > 0 ->
         NewLevel is Level - 1
      |  NewLevel is 0),
      NewParams = [level_limit(NewLevel)|Params2].
find_level(Params, 1, Params).

post_to_all_clients(Event) :-
        oaa_Id(FacId),
        oaa:oaa_solve_local(agent_data(ClientId, ready, _Solvable,_AgentName),
[]),
```

```
      ClientId \== FacId,
      oaa_PostEvent(Event, [address(ClientId), from(FacId)] ),
      fail.
post_to_all_clients(_Event).

    % This is called when length of KSList is > 0.
    %
    % goal_count(GoalId,Goal,Params,EvParams,ToBeCalled,Called,
    %            Responders,Solvers,Answers,NumAnswers)

dispatch_solve_request(RequestingKS, Goal, Params, EvParams, KSList) :-
    new_goal_id(Id),
    % Note that reply (none) overrides parallel_ok (false).  We can't
    % provide parallel_ok (false) if no replies come back from solvers.
    ( memberchk(reply(none), Params) ->
      dispatch_solve_events(KSList, Id, RequestingKS, Goal, Params, EvParams)
    | memberchk(parallel_ok(false), Params) ->
      % Dispatch to one KS; save the rest for later.
      KSList = [FirstKS | Rest],
        assert(goal_count(Id, Goal, Params, EvParams, Rest,
                          [FirstKS], [], [], [], 0)),
      dispatch_solve_event(Id, RequestingKS, Goal, Params, EvParams, FirstKS)
    | otherwise ->
      % Dispatch to all KSs.
      assert(goal_count(Id, Goal, Params, EvParams, [],
                        KSList, [], [], [], 0)),
      dispatch_solve_events(KSList, Id, RequestingKS, Goal, Params, EvParams)
    ).


dispatch_solve_events([], _Id, _RequestingKS, _Goal, _Params, _EvParams).
dispatch_solve_events([SomeKS | Rest], Id, RequestingKS, Goal,
                      Params, EvParams) :-
    dispatch_solve_event(Id, RequestingKS, Goal, Params, EvParams, SomeKS),
    dispatch_solve_events(Rest, Id, RequestingKS, Goal, Params, EvParams).

dispatch_solve_event(Id, RequestingKS, Goal, Params, EvParams, SomeKS) :-
    oaa_Id(SomeKS),
    % That's me, the facilitator.
    !,
    icl_GoalComponents(Goal, _, _, GoalParams),
    append(Params, EvParams, InheritedParams),
    append(GoalParams, InheritedParams, AllParams),
    findall(Goal,
              % InheritedParams here is right, not AllParams:
            oaa:oaa_solve_local(Goal, InheritedParams),
          Solutions),
    ( memberchk(reply(none), AllParams) ->
      true
    | otherwise ->
        oaa_AppDoEvent(

ev_respond_or_post_higher(RequestingKS,SomeKS,Id,Goal,Params,Solutions),
        [])
    ).
dispatch_solve_event(Id, RequestingKS, Goal, Params, _EvParams, SomeKS) :-
    oaa_TraceMsg('~nRouting goal "ev_solve(~p)" to ~p.~n', [Goal, SomeKS]),
```

```
        % ask a sub-agent to try and solve goal.
        %   if solutions are returned, pass them to requestingKS.
        %   otherwise, ask higher blackboard to try and solve goals.
        % note: send ev_solve(id(Id,SomeKS), ...) as a means of insuring
        %   that each ev_solved() trigger is unique and only matches
        %   exactly one response.  We use _SomeKS in the field indicating
        %   which agent actually solved the goal because individual
        %   agents don't necessarily know their internal unique ID #.
    oaa_PostEvent( ev_solve(id(Id,SomeKS), Goal, Params),
                   [address(SomeKS), from(RequestingKS)]),
    ( memberchk(reply(none), Params) ->
        true
    | otherwise ->
          % If time_limit specified in parameters, setup
          % time_trigger to wakeup if solutions hasn't been returned
          % in specified time.
        ( memberchk(time_limit(NSecs), Params) ->
          add_time_check(NSecs, Id, RequestingKS, Goal,Params)
      | true),
      oaa:oaa_add_trigger_local(
        comm,
        event(ev_solved(id(Id,SomeKS), _SomeKS, Goal, P2, Solutions), _),
        ev_respond_or_post_higher(RequestingKS,SomeKS,Id,Goal,P2,Solutions),
        [recurrence(when), on(receive)])
    ).

% return_solutions(+RequestingKS, +Responder, +Id, +Goal, +P, +NewSolutions).
% Having just received solutions from a Responder, take the appropriate action.
%
% Even though the Responder has returned copies of the goal and params,
% we don't need them because we have a local copy in goal_count.
%
% @@DLM: Unresolved question about streaming: Should we stream the
% responses with 0 solutions?  [My thinking is "yes".]
return_solutions(RequestingKS, Responder, Id, _Goal, _P, NewSolutions) :-
    % ToBeCalled lists solvers not yet called.  PrevCalled lists
    % the called solvers that have yet to respond.
    retract(goal_count(Id, Goal, Params, EvParams,
                       ToBeCalled, PrevCalled, PrevResponders,
                       PrevSolvers, PrevSolutions, PrevNumSol)),
    !,
      % Take Responder out of the called list:
    ( selectchk(Responder, PrevCalled, Called) ->
        true
    | otherwise ->
      format('ERROR: Inappropriate ev_solved event received:~n', []),
      format('  ~w ~w ~w ~w~n', [RequestingKS, Responder, Id, Goal]),
      Called = PrevCalled
    ),
      % and put him into the responder list:
    append(PrevResponders, [Responder], Responders),
      % The solvers are just the responders that succeeded:
    ( NewSolutions = [] ->
      NewSolvers = []
    | otherwise ->
      NewSolvers = [Responder]
    ),
```

```
append(PrevSolvers, NewSolvers, Solvers),
append(PrevSolutions, NewSolutions, Solutions),
length(NewSolutions, NewNumSol),
NumSol is PrevNumSol + NewNumSol,

  % This case means that either: (1) we've gotten responses from all
  % solvers; and/or (2) we have reached the desired number of solutions.
  % By not saving goal_count, we ensure that any additional returned
  % solutions are ignored:
( ((ToBeCalled == [], Called == []) ;
   (memberchk(solution_limit(Limit), Params), NumSol >= Limit)) ->
      % This test is a place-holder; streaming not yet official:
    ( memberchk(reply(streaming), Params) ->
        Return = ev_reply_solved([Responder], NewSolvers, Goal, Params,
                                 NewSolutions)
  | otherwise ->
        Return = ev_reply_solved(Responders, Solvers, Goal, Params,
                                 Solutions)
  ),
  Save = false
  % This case happens with parallel_ok(false):
| ToBeCalled = [Next | Rest] ->
  dispatch_solve_event(Id, RequestingKS, Goal, Params, EvParams, Next),
      % This test is a place-holder; streaming not yet official:
    ( memberchk(reply(streaming), Params) ->
        Return = ev_reply_solved([Responder], NewSolvers, Goal, Params,
                                 NewSolutions),
        Save = goal_count(Id, Goal, Params, EvParams,
                          Rest, [Next|Called], [], [], [], NumSol)
    | otherwise ->
      Return = false,
        Save = goal_count(Id, Goal, Params, EvParams,
                          Rest, [Next|Called], Responders, Solvers,
                     Solutions, NumSol)
    )
  % Still waiting for some called solvers to respond:
| Called = [_ | _] ->
      % This test is a place-holder; streaming not yet official:
    ( memberchk(reply(streaming), Params) ->
        Return = ev_reply_solved([Responder], NewSolvers, Goal, Params,
                                 NewSolutions),
        Save = goal_count(Id, Goal, Params, EvParams,
                          ToBeCalled, Called, [], [], [], NumSol)
    | otherwise ->
      Return = false,
        Save = goal_count(Id, Goal, Params, EvParams,
                          ToBeCalled, Called, Responders, Solvers,
                     Solutions, NumSol)
    )
),
( Save == false ->
    true
| otherwise ->
   assert(Save)
),
( Return == false ->
    true
```

```
    | otherwise ->
        oaa_TraceMsg('~nRouting answers back to ~p:~n   ~p~n',
                     [RequestingKS,Return]),
        oaa_PostEvent(Return, [address(RequestingKS)])
    ).
return_solutions(_RequestingKS, _Responder, _Id, _Goal, _P, _NewSolutions).

dispatch_update_request(RequestingKS, Mode, Clause, Params, []) :-
    % No agents able to perform the requested update:
    !,
    ( memberchk(reply(none), Params) ->
      true
    | otherwise ->
      Event = ev_reply_updated(Mode, Clause, Params, [], []),
      oaa_PostEvent(Event, [address(RequestingKS)])
    ).
dispatch_update_request(RequestingKS, Mode, Clause, Params, KSList) :-
    new_goal_id(Id),
    length(KSList,NumKSsForGoal),
    % if more than one KS can solve the goal, remember so that
    % we can collect answers from all of them later
    ( NumKSsForGoal > 1 ->
        assert(update_count(Id, NumKSsForGoal, [], []))
    | otherwise ->
        true
    ),
    member(SomeKS, KSList),    % backtrack over all KSs.
    dispatch_update_event(Id, RequestingKS, Mode, Clause, Params, SomeKS),
    fail.
dispatch_update_request(_RequestingKS, _Mode, _Clause, _Params, _KSList).

dispatch_update_event(Id, RequestingKS, Mode, Clause, Params, SomeKS) :-
    oaa_Id(SomeKS),
    % That's me, the facilitator.
    !,
    ( Mode == add ->
        Functor = oaa_add_data_local
    | Mode == replace ->
        Functor = oaa_replace_data_local
    | otherwise ->
        Functor = oaa_remove_data_local
    ),
    append(Params, [from(RequestingKS)], AllParams),
    Goal =.. [Functor, Clause, AllParams],
    ( call(oaa:Goal) ->
        Updaters = [SomeKS]
    | otherwise ->
        Updaters = []
    ),
    ( memberchk(reply(none), Params) ->
      true
    | otherwise ->
        % Params must be returned here (not AllParams):
        return_update(RequestingKS,Mode,SomeKS,Id, Clause,Params,Updaters)
    ).
dispatch_update_event(Id, RequestingKS, Mode, Clause, Params, SomeKS) :-
    oaa_TraceMsg('~nRouting request "ev_update(~p, ~p, ~p)" to ~p.~n',
```

```
            [Mode, Clause, Params, SomeKS]),
    append(Params, [from(RequestingKS)], AllParams),
    oaa_PostEvent(
             ev_update(id(Id,SomeKS), Mode, Clause, AllParams),
             [address(SomeKS)]),
    ( memberchk(reply(none), Params) ->
      true
    | otherwise ->
      % TBD: Do we want to set a time trigger here?
      oaa:oaa_add_trigger_local(
          comm,
          event(ev_updated(id(Id,SomeKS), _Mode, _Clause, _P2, Updaters), _),
            % Params must be returned here (not AllParams):
          ev_return_update(RequestingKS,Mode,SomeKS,Id,
                           Clause,Params,Updaters),
          [recurrence(when), on(receive)])
    ).


% Returns, to requesting KS, the addresses of all agents (including
% facilitator if appropriate), that attempted (NewKSs) and that actually
% satisfied (Updaters) an update request.
%
% NewUpdaters is always either [], or a singleton list.
%
% Possible values for Mode: add, remove, replace.
%
% Note: Params must be returned in ev_reply_updated, so it must be
% unifiable with the params embedded in the requesting event (ev_post_event).

return_update(RequestingKS, Mode, Responder, Id, Clause, Params,
              NewUpdaters) :-
      retract(update_count(Id, AgentsLeft, PrevKSs, PrevUpdaters)),
      append(PrevUpdaters, NewUpdaters, Updaters),
      append(PrevKSs, [Responder], NewKSs),
      ( AgentsLeft > 1 ->
         NewAgentsLeft is AgentsLeft - 1,
         assert(update_count(Id, NewAgentsLeft, NewKSs, Updaters))
      | otherwise ->
          oaa_TraceMsg('~nRouting updaters back to ~p:~n   ~p~n',
                   [RequestingKS,Updaters]),
          Event = ev_reply_updated(Mode, Clause, Params, NewKSs, Updaters),
          oaa_PostEvent(Event, [address(RequestingKS)])
      ), !.
return_update(RequestingKS, Mode, Responder, _Id, Clause, Params, Updaters) :-
    oaa_TraceMsg('~nRouting updaters back to ~p:~n   ~p~n',
             [RequestingKS,Updaters]),
    Event = ev_reply_updated(Mode, Clause, Params, [Responder], Updaters),
    oaa_PostEvent(Event, [address(RequestingKS)]).

    % No agents able to install this trigger:
dispatch_trigger_request(RKS, Mode, Type, Condition, Action, Params, []) :-
    !,
    ( memberchk(reply(none), Params) ->
      true
    | otherwise ->
      Event = ev_reply_trigger_updated(Mode, Type, Condition, Action, Params,
```

```
                                [], []),
      oaa_PostEvent(Event, [address(RKS)])
    ).
dispatch_trigger_request(RKS, Mode, Type, Condition, Action, Params, KSList) :-
    new_goal_id(Id),
    length(KSList,NumKSsForGoal),
    % if more than one KS can solve the goal, remember so that
    % we can collect answers from all of them later
    ( NumKSsForGoal > 1 ->
        assert(update_count(Id, NumKSsForGoal, [], []))
    | otherwise ->
        true
    ),
    member(SomeKS, KSList),     % backtrack over all KSs.
    dispatch_trigger_event(Id, RKS, Mode, Type, Condition, Action, Params,
SomeKS),
    fail.
dispatch_trigger_request(_RKS, _Mode, _Type, _Condition, _Action, _Params,
                    _KSList).

dispatch_trigger_event(Id, RKS, Mode, Type, Condition, Action, Params,
                    SomeKS) :-
    oaa_Id(SomeKS),
    % That's me, the facilitator.
    !,
    ( Mode == add ->
        Functor = oaa_add_trigger_local
    | otherwise ->
        Functor = oaa_remove_trigger_local
    ),
    Goal =.. [Functor, Type, Condition, Action, Params],
    ( call(oaa:Goal) ->
        Updaters = [SomeKS]
    | otherwise ->
        Updaters = []
    ),
    ( memberchk(reply(none), Params) ->
      true
    | otherwise ->
        return_trigger_update(RKS, Mode, SomeKS, Id, Type,
                              Condition, Action, Params, Updaters)
    ).
dispatch_trigger_event(Id, RKS, Mode, Type, Condition, Action, Params,
                    SomeKS) :-
    oaa_TraceMsg('~nRouting request~n  ev_update_trigger(~p, ~p, ~p, ~p, ~p)~nto
~p.~n',
                 [Mode, Type, Condition, Action, Params, SomeKS]),
    oaa_PostEvent(
       ev_update_trigger(id(Id,SomeKS), Mode, Type, Condition, Action, Params),
        [address(SomeKS), from(RKS)]),
    ( memberchk(reply(none), Params) ->
      true
    | otherwise ->
      % TBD: Do we want to set a time trigger here?
      oaa:oaa_add_trigger_local(
          comm,
```

```
          event(ev_trigger_updated(id(Id,SomeKS), _Mode, _Type, _Condition,
_Action, P2, Updaters), _),
          ev_return_trigger_update(RKS,Mode,SomeKS,Id,
                                   Type,Condition,Action,P2,Updaters),
          [recurrence(when), on(receive)])
    ).


% Returns, to requesting KS, the addresses of all agents (including
% facilitator if appropriate), that attempted (NewKSs) and that actually
% satisfied (Updaters) a trigger update request.
%
% NewUpdaters is always either [], or a singleton list.
%
% Possible values for Mode: add, remove.

return_trigger_update(RequestingKS, Mode, Responder, Id,
                      Type, Condition, Action, Params, NewUpdaters) :-
      retract(update_count(Id, AgentsLeft, PrevKSs, PrevUpdaters)),
      append(PrevUpdaters, NewUpdaters, Updaters),
      append(PrevKSs, [Responder], NewKSs),
      ( AgentsLeft > 1 ->
         NewAgentsLeft is AgentsLeft - 1,
         assert(update_count(Id, NewAgentsLeft, NewKSs, Updaters))
      | otherwise ->
          Event = ev_reply_trigger_updated(Mode,Type,Condition,Action,
                                           Params, NewKSs, Updaters),
          oaa_PostEvent(Event, [address(RequestingKS)])
      ), !.
return_trigger_update(RequestingKS, Mode, Responder, _Id,
                      Type, Condition, Action, Params, Updaters) :-
    Event = ev_reply_trigger_updated(Mode, Type, Condition, Action,
                                     Params, [Responder], Updaters),
    oaa_PostEvent(Event, [address(RequestingKS)]).

    % unify_params(+OrigParams, +Params, -UnifiedParams).
    %
    % There is now (970219) a requirement that the params returned in
    % a ev_solved or ev_solved_by_bb event must be unifiable with the original
    % params from the corresponding solve request.  In some situations*, the
    % Params returned to the facilitator by a solver may not unify with
    % the OrigParams, but may contain individual elements with variables
    % instantiated by the solver.  This pred. can be used to save these
    % instantiations.
    %
    % *Such as, when find_level has been used to create a new params list.
unify_params([], _Params, []).
unify_params([OrigParam | Rest], Params, [OrigParam | UnifiedRest]) :-
    ( memberchk(OrigParam, Params) | true ),
    !,
    unify_params(Rest, Params, UnifiedRest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% These are extremely simple predicates for maintaining com_connection_info/5,
% which keeps info about the agents to which this agent currently has
% a communications channel.
```

```
add_connected(Id, Connection) :-
   assert(com:com_connection_info(Id, unknown, child,
                        [connection(Connection),oaa_id(Id)], connected)).

update_connected(Id, AddInfo) :-
   com_AddInfo(Id, AddInfo).

% remove_connected(+Id).
remove_connected(Id) :-
   retractall(com:com_connection_info(Id, _, _, _, _)).

% if the time_limit(NSec) parameter is sent, install wakeup on server
% to indicate the request has failed if not achieved in the correct time.
add_time_check(NSecs, Id, RequestingKS, Goal,Params) :-
   (time_limit_trigger(Id,_When, RequestingKS,_Goal,_Params) ->
      true   % already added for this goal request
   |
      tcp_now(Now),
      tcp_time_plus(Now,NSecs,Soon),
      tcp_schedule_wakeup(Soon, time_limit(Id)),
      assert(time_limit_trigger(Id,Soon,RequestingKS,Goal,Params)),
      oaa_TraceMsg('~nTime limit check added for ~p~n',[Goal])
   ), !.


% if solutions are returned before a time_limit_trigger has expired,
%  remove the trigger.
cancel_time_check(Id) :-
   retract(time_limit_trigger(Id,When,_RequestingKS,Goal,_Params)),
   tcp_cancel_wakeup(When,time_limit(Id)),
   oaa_TraceMsg('~nTime limit check removed because solution returned.~n
~p~n',
      [Goal]), !.
cancel_time_check(_Id).



/* Generates a unique ID for a goal.          */
/* ID's should be unique across blackboards*/
/*   which is why we use the KSName prefix */
/* Goal counters are used to make sure the */
/* solution really matches the query.         */

new_goal_id(NewId) :-
      oaa_Name(KSName),
      concat(KSName, '_', Tmp),
      gensym(Tmp, NewId).

% Returns a list containing Num new goal ids.

new_goal_ids(Num, [NewId | RestIds]) :-
    Num > 0,
    !,
    new_goal_id(NewId),
    NewNum is Num - 1,
    new_goal_ids(NewNum, RestIds).
new_goal_ids(_Num, []).
```

```
start :-
    runtime_entry(start).

runtime_entry(start) :-
    initial_solvables(Solvables),
    com_ListenAt(incoming, CInfo),
    format('Listening at ~p~n~n', [CInfo]),
    oaa_RegisterCallback(app_do_event, user:oaa_AppDoEvent),
    oaa_Register(incoming, 'root', Solvables),
    on_exception(_, oaa_AppInit, true),
    oaa_MainLoop(true).


runtime_entry(abort) :- !.
%     format('Closing all connections...~n',[]),
%     close_all_connections.

% If the Facilitator is killed (ctrl-c) before disconnecting
% all clients, it will not free the port.
% This code is an attempt to fix this problem, but it doesn't
% help.  Why not???
% close_all_connections :-
%     tcp_connected(X,Y),
%     tcp_destroy_listener(Y),
%     tcp_shutdown(X),
%     fail.
% close_all_connections :-
%     tcp_reset, fail.
```